



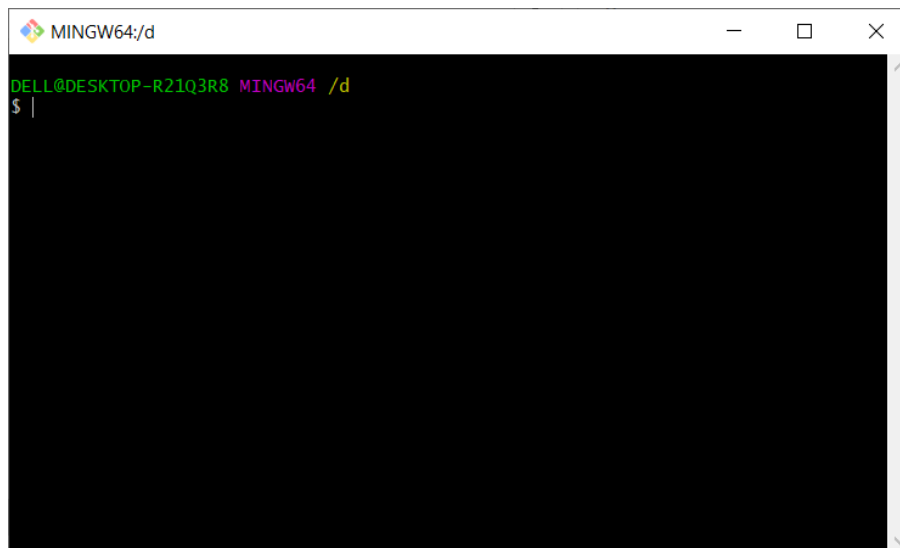
# Manual de Git y GitHub Desktop

Ing. Luis Guillermo Molero Suárez



## ¿Qué es Git?

Es un sistema de control de versiones, es distribuido, es decir que múltiples personas pueden trabajar en equipo, es open source y también se adapta a todo tipo de proyectos desde pequeños hasta grandes, además, se pueden fusionar archivos, guarda una línea de tiempo a lo largo de todo el proyecto. Maneja una interfaz tipo Bash. **GIT, es el software de control de versiones en el que se basa GitHub.**



Sitio de descarga: <https://git-scm.com/downloads>

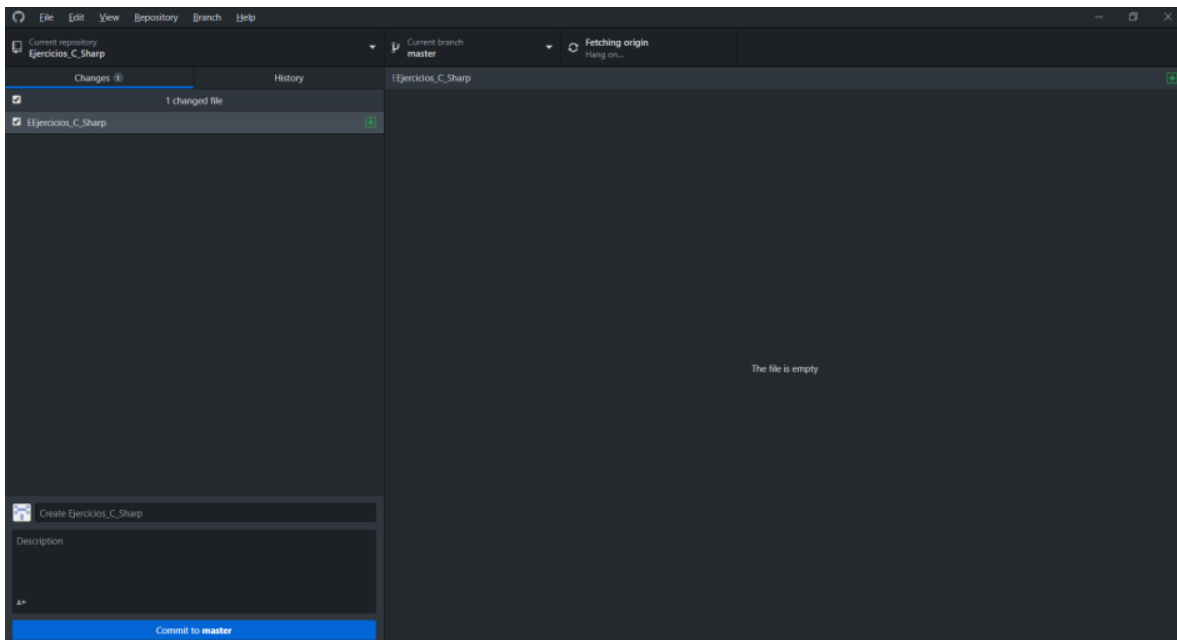
Como instalar Git: <https://www.youtube.com/watch?v=ExdLS6IZaAY>



## ¿Qué es GitHub?

A diferencia de Git, GitHub es un sitio web y un servicio en la nube que ayuda a los desarrolladores a almacenar y administrar su código, al igual que llevar un registro y control de cualquier cambio sobre este código. En otras palabras, es una plataforma de desarrollo colaborativo, o también llamada la red social de los desarrolladores donde se alojan los repositorios, el código se almacena de forma pública pero se puede hacer privado con una cuenta de pago.

La interfaz de **GitHub Desktop** es bastante fácil de usar para el desarrollador novato que quiera aprovechar las ventajas del Git. Sin GitHub Desktop, usar un Git generalmente requiere de un poco más de conocimientos de tecnología y uso de una línea de comando (Bash).



Sitio de descarga: <https://desktop.github.com/>

Como instalar GitHub Desktop: <https://www.youtube.com/watch?v=tn6tloweTUs>

## ¿Qué es Control de Versiones?

El control de versiones ayuda a los desarrolladores a llevar un registro y administrar cualquier cambio en el código del proyecto de software. A medida que crece este proyecto, la versión de control se vuelve esencial.

Con la bifurcación, un desarrollador duplica parte del código fuente (llamado repositorio). Este desarrollador, luego puede, de forma segura, hacer cambios a esa parte del código, sin afectar al resto del proyecto.



Luego, una vez que el desarrollador logre que su parte del código funcione de forma apropiada, esta persona podría fusionar este código al código fuente principal para hacerlo oficial. Todos estos cambios luego son registrados y pueden ser revertidos si es necesario.

Documentación de GitHub: <https://docs.github.com/es/github>

Documentación Git: <https://git-scm.com/book/es/v2>

## Creación de una cuenta

Lo primero que necesitas es una cuenta de usuario gratuita. Simplemente visita <https://github.com>, elige un nombre de usuario que no esté ya en uso, proporciona un correo y una contraseña, y pulsa el botón verde grande “Sign up for GitHub”.

Formulario de registro de GitHub. El formulario tiene un fondo negro y tres campos de entrada blancos: 'Pick a username', 'Your email' y 'Create a password'. Debajo del campo de contraseña, hay un texto que dice: 'Use at least one lowercase letter, one numeral, and seven characters.' En la parte inferior del formulario hay un botón verde con el texto 'Sign up for GitHub'.

Lo siguiente que verás es la página de precios para planes mejores, pero lo puedes ignorar por el momento. GitHub te enviará un correo para verificar la dirección que les has dado. Confirmar la dirección ahora, es bastante importante (como veremos después).

Para ampliar esta información: <https://n9.cl/nqu9>

## Crear un repositorio

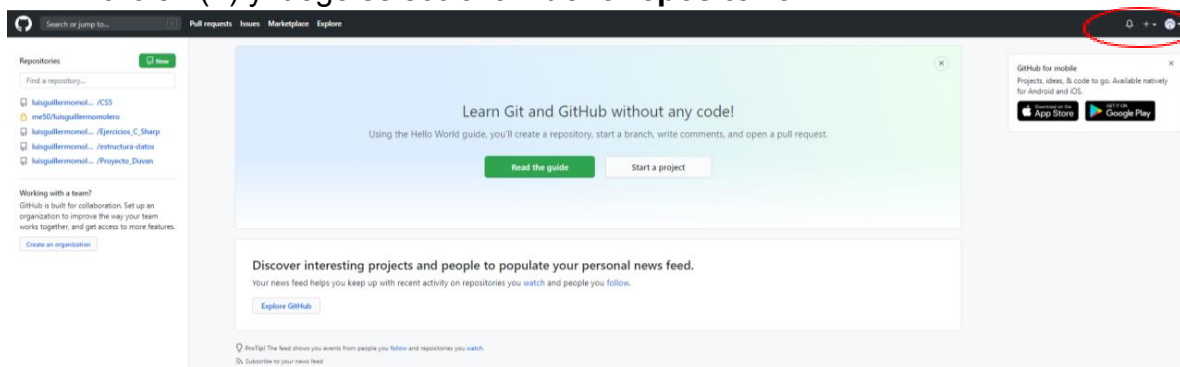
Un **repositorio** se usa generalmente para organizar un solo proyecto. Los repositorios pueden contener carpetas y archivos, imágenes, videos, hojas de cálculo y conjuntos de datos, cualquier cosa que su proyecto necesite. Recomendamos incluir un archivo README o un archivo con información sobre su proyecto. GitHub facilita agregar uno al mismo tiempo que crea su nuevo repositorio. *También ofrece otras opciones comunes, como un archivo de licencia.*



Tomaremos como ejemplo didáctico, la creación de un repositorio para los ejercicios de estructuras de datos.

## Paso 1. Crear un nuevo repositorio

1. En la esquina superior derecha, junto a tu avatar o icono de identidad, haz clic en (+) y luego selecciona **Nuevo repositorio**.




2. Nombra tu repositorio `estructura-datos`
3. Escribe una breve descripción.

### Create a new repository


A repository contains all project files, including the revision history. Already have a project repository elsewhere?


[Import a repository.](#)

Owner \* luisguillermomolero / Repository name \* estructura-datos 

Great repository names are short and memorable. Need inspiration? How about [literate-funicular](#)?


Description (optional) Repositorio ejercicios materia Estructura de Datos


☒  **Public**  
Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**  
You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

☐ **Initialize this repository with a README**  
This will let you immediately clone the repository to your computer.

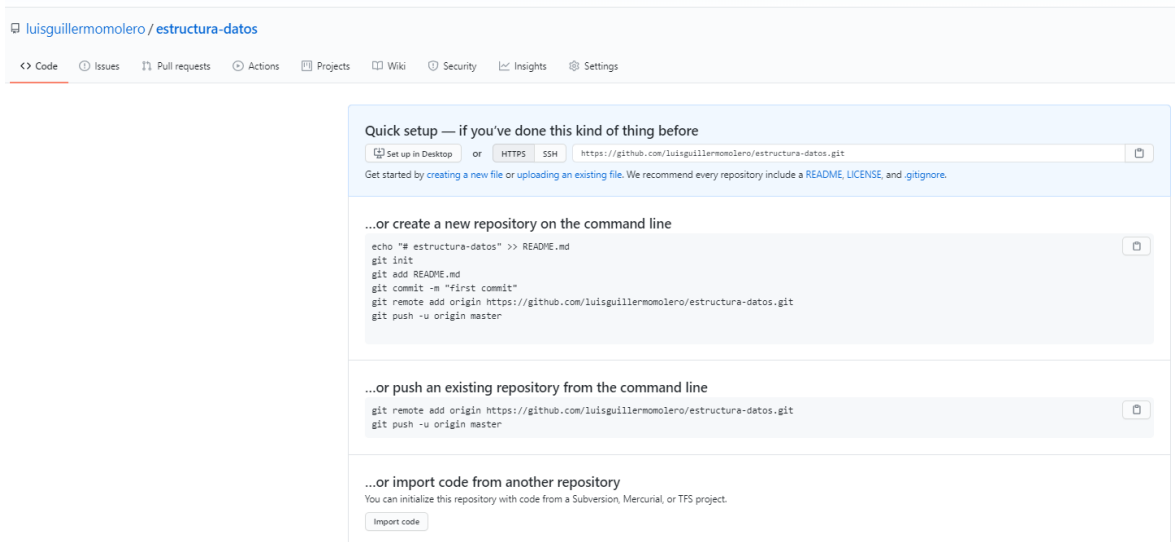
Add .gitignore: None Add a license: None 



4. Haga clic en **Crear repositorio**.

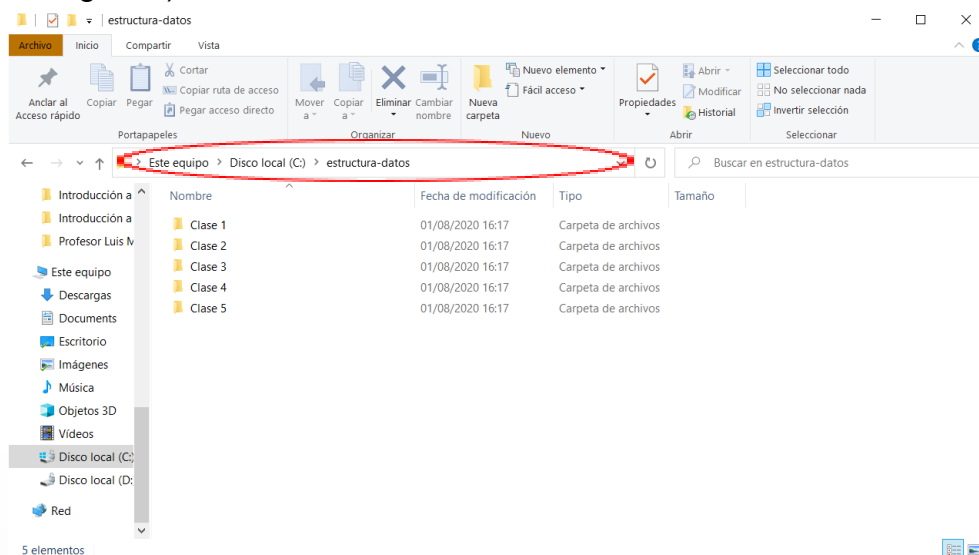


## 5. **Guarda** estos valores debido a que luego lo necesitarás para subir tus aplicaciones desde el *Bash* de **Git** a este **GitHub**



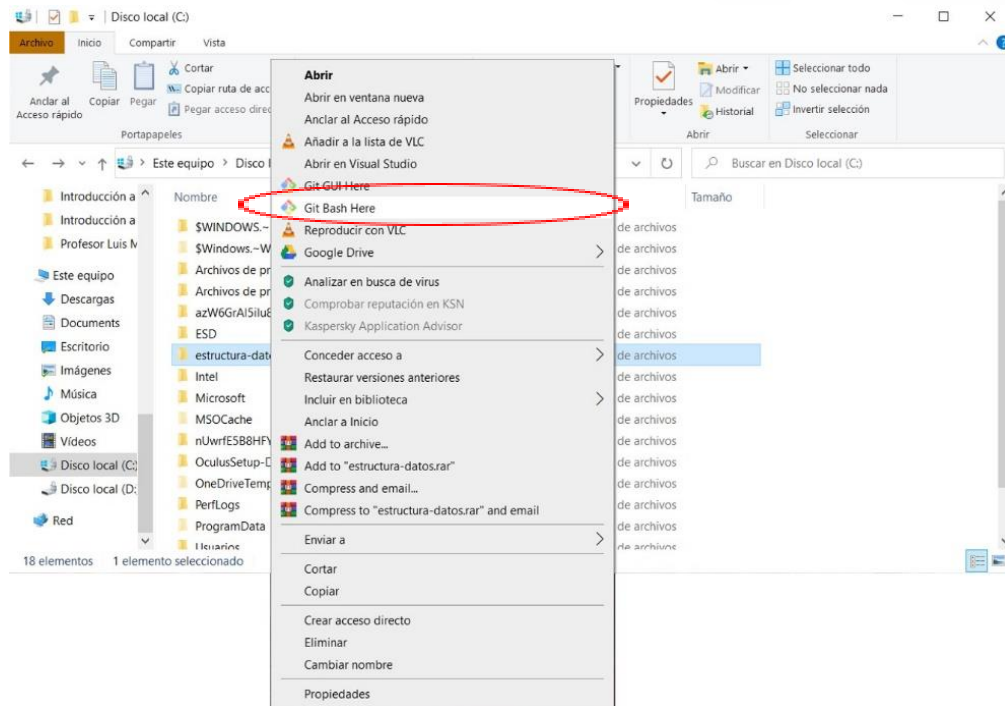
## Paso 2. Subir nuestros proyectos a Github

1. Crear una carpeta en el disco duro C:\ (ruta: C:\estructura-datos) para guardar todos los ejercicios resueltos de la materia estructura de datos.
2. Crear una carpeta dentro de C:\estructura-datos para cada una de las clases (de hacerse de otro modo no serán evaluadas los ejercicios entregados)

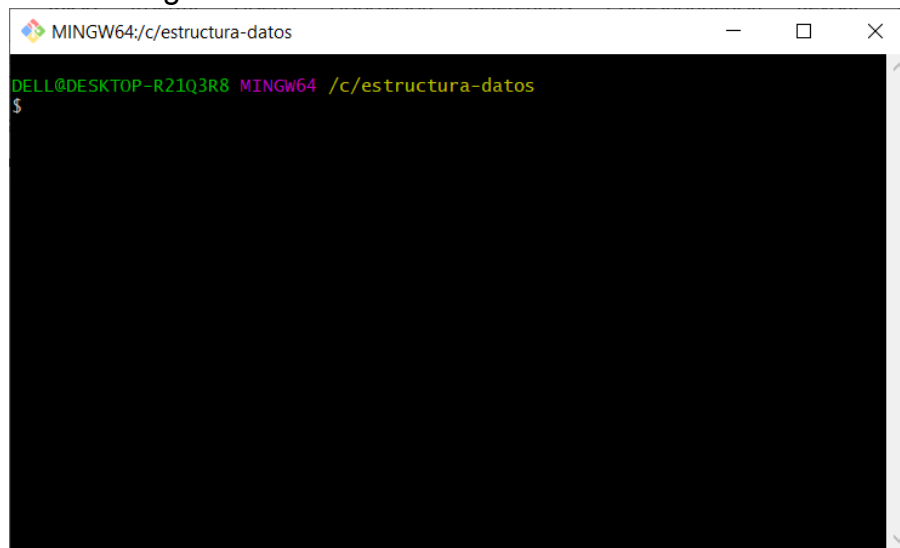




- Una vez creada las carpetas de cada clase y luego de haber instalado **Git y GitHub Desktop**, hacer  clic derecho  sobre la carpeta “**estructura-datos**” y luego sobre la opción “**Git Bash Here**”.



Se abrirá la siguiente consola Bash



- Escribir la siguiente línea de comando para iniciar el **Git** en esa carpeta.  
**(ESTO SOLO SE HACE UNA VEZ)**



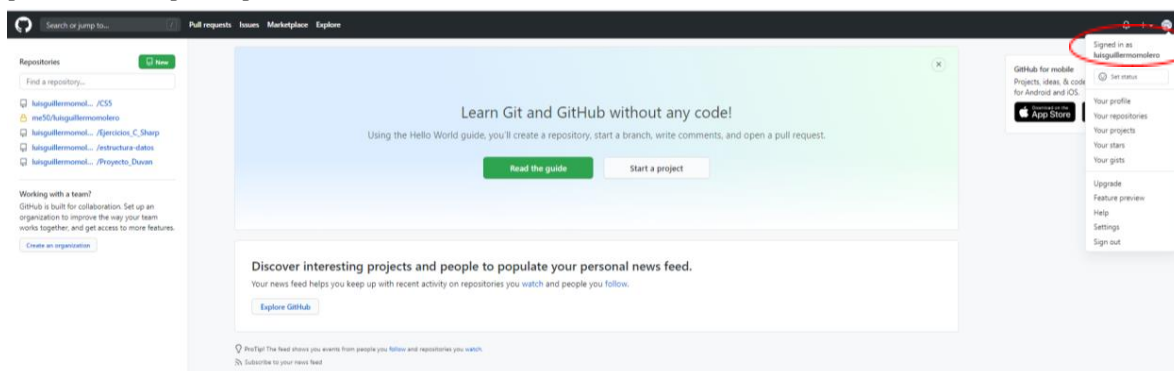
git init

```
MINGW64:/c/estructura-datos
DELL@DESKTOP-R21Q3R8 MINGW64 /c/estructura-datos
$ git init
Initialized empty Git repository in C:/estructura-datos/.git/
DELL@DESKTOP-R21Q3R8 MINGW64 /c/estructura-datos (master)
$
```

A continuación, todo lo que agregue dentro de esa carpeta estará dentro del Git y se podrá sincronizar en la nube. Asimismo, dentro de esa carpeta se creará una carpeta llamada .git que está oculta.

5. Escribir la siguiente línea de comandos para establecer la configuración de nombre de usuario / correo electrónico específica del repositorio: **(ESTO SOLO SE HACE UNA VEZ)**. Debes usar el mismo correo que utilizaste al crear tu cuenta en Github.

git config --global user.name "tu nombre de usuario en Github"





```
git config --global user.email "tu correo en Github"
```

```
MINGW64:/c/estructura-datos

DELL@DESKTOP-R21Q3R8 MINGW64 /c/estructura-datos
$ git init
Initialized empty Git repository in C:/estructura-datos/.git/

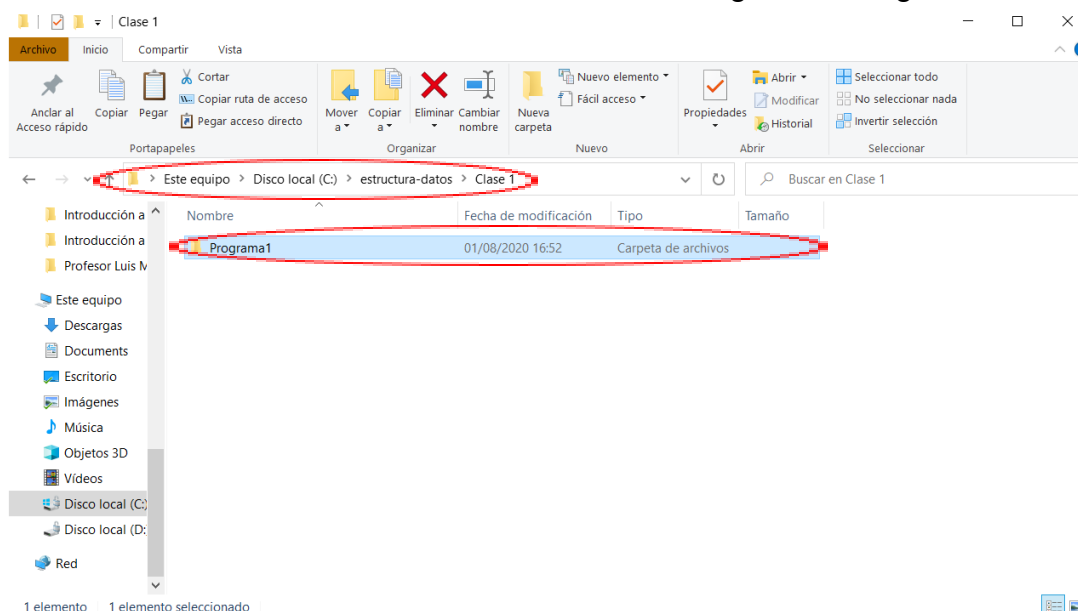
DELL@DESKTOP-R21Q3R8 MINGW64 /c/estructura-datos (master)
$ git config --global user.name "luisguillermomolero"

DELL@DESKTOP-R21Q3R8 MINGW64 /c/estructura-datos (master)
$

DELL@DESKTOP-R21Q3R8 MINGW64 /c/estructura-datos (master)
$ git config --global user.email "luisguillermomolero@gmail.com"

DELL@DESKTOP-R21Q3R8 MINGW64 /c/estructura-datos (master)
$
```

Supongamos que en esta primera entrega ud. va a subir el primer programa realizado como tarea, como se muestra en la siguiente imagen:



6. Para subir ese primer ejercicio resuelto, debes repetir solo el paso 3 y cuando aparezca el Bash de GIT escribir la siguiente línea de comando: (ESTO LO DEBES HACER CADA VEZ QUE MODIFIQUES ALGÚN EJERCICIO O HAGAS UN EJERCICIO NUEVO)





git add .

```
MINGW64/c/estructura-datos

DELL@DESKTOP-R21Q3R8 MINGW64 /c/estructura-datos (master)
$ git add .

DELL@DESKTOP-R21Q3R8 MINGW64 /c/estructura-datos (master)
$
```

**git add** : Lleva el control de los archivos que se agregan luego de escribir ese comando y **git add .** sirve para agregar todos los archivos modificados después del primer envío (commit).

7. Una vez ejecutado el `git add .` ejecutamos la siguiente línea de comando para validar el estado actual de nuestro **Git**.

git status

```
MINGW64/c/estructura-datos

DELL@DESKTOP-R21Q3R8 MINGW64 /c/estructura-datos (master)
$ git add .
DELL@DESKTOP-R21Q3R8 MINGW64 /c/estructura-datos (master)
$ git status
On branch master
No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   Clase 1/Programa1/DesignTimeBuild/.dtb.cache.v2
        new file:   Clase 1/Programa1/vs/Programa1/v15/.csc
        new file:   Clase 1/Programa1/Programa1.csproj
        new file:   Clase 1/Programa1/Programa1.sln
        new file:   Clase 1/Programa1/bin/Debug/netcoreapp3.1/Programa1.deps.json
        new file:   Clase 1/Programa1/bin/Debug/netcoreapp3.1/Programa1.dll
        new file:   Clase 1/Programa1/bin/Debug/netcoreapp3.1/Programa1.exe
        new file:   Clase 1/Programa1/bin/Debug/netcoreapp3.1/Programa1.pdb
        new file:   Clase 1/Programa1/bin/Debug/netcoreapp3.1/Programa1.runtimeconfig.dev.json
        new file:   Clase 1/Programa1/bin/Debug/netcoreapp3.1/Programa1.runtimeconfig.json
        new file:   Clase 1/Programa1/obj/Debug/netcoreapp3.1/Programa1.AssemblyInfo.cs
        new file:   Clase 1/Programa1/obj/Debug/netcoreapp3.1/Programa1.AssemblyInfo.cs.cache
        new file:   Clase 1/Programa1/obj/Debug/netcoreapp3.1/Programa1.assets.cache
        new file:   Clase 1/Programa1/obj/Debug/netcoreapp3.1/Programa1.csproj.C
        new file:   Clase 1/Programa1/obj/Debug/netcoreapp3.1/Programa1.csproj.F
        new file:   Clase 1/Programa1/obj/Debug/netcoreapp3.1/Programa1.csproj.assets.cache
        new file:   Clase 1/Programa1/obj/Debug/netcoreapp3.1/Programa1.dll
        new file:   Clase 1/Programa1/obj/Debug/netcoreapp3.1/Programa1.exe
        new file:   Clase 1/Programa1/obj/Debug/netcoreapp3.1/Programa1.genruntimeconfig.cache
        new file:   Clase 1/Programa1/obj/Debug/netcoreapp3.1/Programa1.pdb
        new file:   Clase 1/Programa1/obj/Programa1.csproj.nuget.dgspec.json
        new file:   Clase 1/Programa1/obj/Programa1.csproj.nuget.g.props
        new file:   Clase 1/Programa1/obj/Programa1.csproj.nuget.g.targets
        new file:   Clase 1/Programa1/project.assets.json
        new file:   Clase 1/Programa1/project.nuget.cache

DELL@DESKTOP-R21Q3R8 MINGW64 /c/estructura-datos (master)
$
```

El comando `git status` te mostrará los diferentes estados de los archivos en tu directorio de trabajo y área de ensayo. Qué archivos están modificados y sin seguimiento y cuáles con seguimiento pero no



confirmados aún. En su forma normal, también te mostrará algunos consejos básicos sobre cómo mover archivos entre estas etapas.

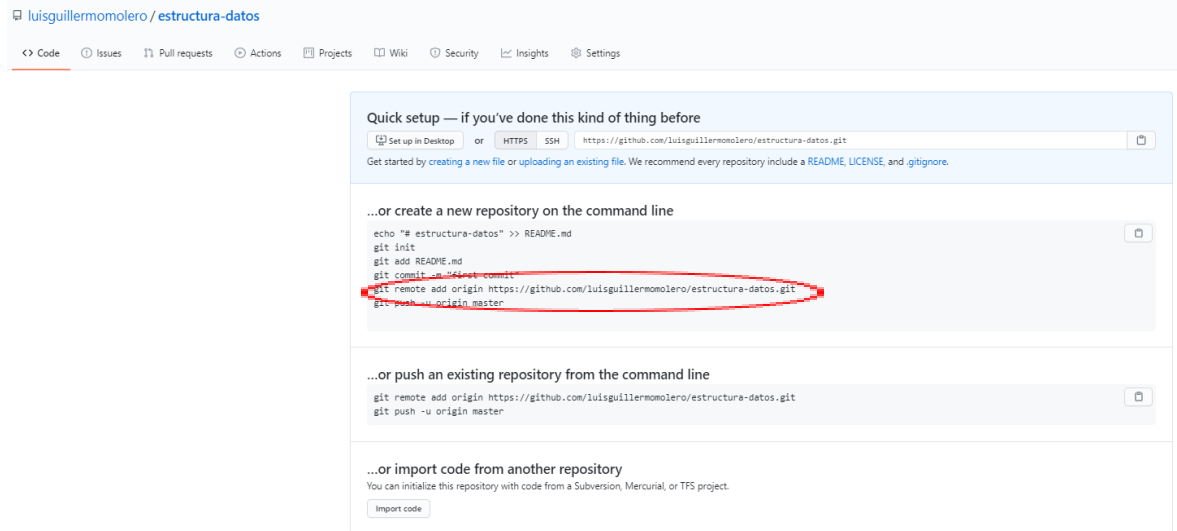
- Una vez ejecutado el comando `git status` ejecutamos la siguiente línea de comando para instanciar los cambios preparados en ese momento.  
`git commit -m "mensaje"`

ejemplo: `git commit -m "primera actualizacion"`

```
MINGW64/c/estructura-datos
git commit -m "primera actualizacion"
[detached HEAD 8210388] primera actualizacion
create mode 100644 .vs/Programal/DesignTimeBuild/dtbcache.v2
create mode 100644 .vs/Programal/vib/suo
create mode 100644 Programal.csproj
create mode 100644 Programal.sln
create mode 100644 Programal.deps.json
create mode 100644 Programal/bin/Debug/netcoreapp3.1/Programal.dll
create mode 100644 Programal/bin/Debug/netcoreapp3.1/Programal.exe
create mode 100644 Programal/bin/Debug/netcoreapp3.1/Programal.pdb
create mode 100644 Programal/bin/Debug/netcoreapp3.1/Programal.runtimeconfig.dev.json
create mode 100644 Programal/bin/Debug/netcoreapp3.1/Programal.runtimeconfig.json
create mode 100644 Programal/obj/Debug/netcoreapp3.1/Programal.AssemblyInfo.cs
create mode 100644 Programal/obj/Debug/netcoreapp3.1/Programal.assets.cache
create mode 100644 Programal/obj/Debug/netcoreapp3.1/Programal.csproj.AssemblyReference.cache
create mode 100644 Programal/obj/Debug/netcoreapp3.1/Programal.dll
create mode 100644 Programal/obj/Debug/netcoreapp3.1/Programal.exe
create mode 100644 Programal/obj/Debug/netcoreapp3.1/Programal.genruntimeconfig.cache
create mode 100644 Programal/obj/Programal.csproj.nuget.dgspec.json
create mode 100644 Programal/obj/Programal.csproj.nuget.g.props
create mode 100644 Programal/obj/Programal.csproj.nuget.g.targets
create mode 100644 Programal/project.assets.json
create mode 100644 Programal/project.nuget.cache
git commit -m "primera actualizacion"
[detached HEAD 8210388] primera actualizacion
MINGW64/c/estructura-datos (master)
```

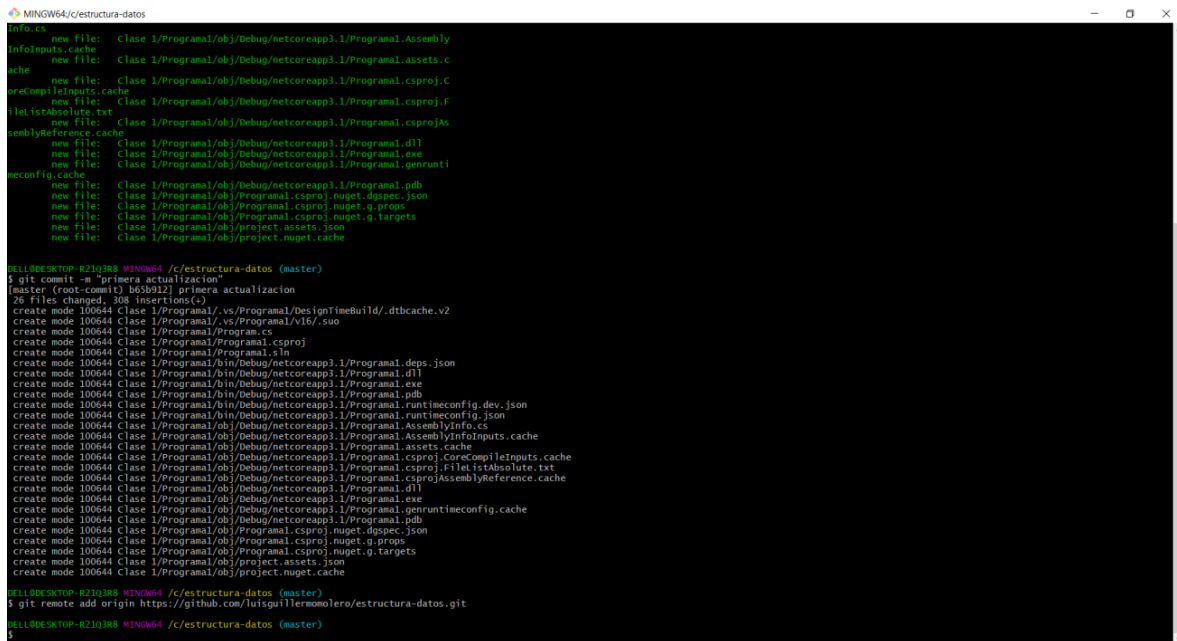
**git commit** : Este comando indica que esta lista alguna funcionalidad para que sea una versión del código. Este comando se repite cada vez que se cambia el código.

- Una vez ejecutado el `git commit` ubicamos los datos que guardamos en el sub-paso 5 (Guarda estos valores debido a que luego lo necesitarás para subir tus aplicaciones desde el **Bash** de **Git** a este **GitHub**) de nuestro Paso 1. (**Paso 1. Crear un nuevo repositorio**)



10. Ejecutamos la siguiente línea de comando que tomamos del paso agregar un nuevo control remoto.

```
git remote add origin https://github.com/luisguillermomolero/estructura-datos.git
```



11. Finalmente, luego de ejecutar la línea de comando anterior, ejecutamos un `git push` para subir tus cambios locales a tu repositorio en línea.

```
git push --set-upstream origin master
```

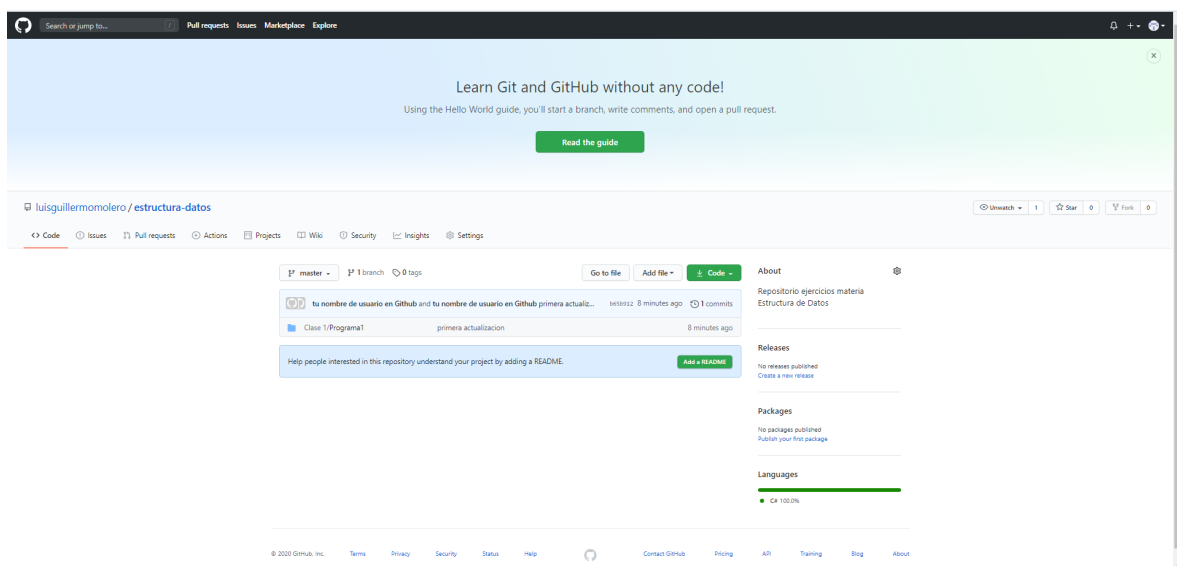


```
MINGW64/c/estructura-datos

DELL@DESKTOP-R21Q3R8 MINGW64 /c/estructura-datos (master)
$ git push --set-upstream origin master
Enumerating objects: 37, done.
Counting objects: 100% (37/37), done.
Delta compression using up to 8 threads
Compressing objects: 100% (26/26), done.
Writing objects: 100% (37/37), 92.59 KiB | 5.79 MiB/s, done.
Total 37 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/luisguillermomolero/estructura-datos.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.

DELL@DESKTOP-R21Q3R8 MINGW64 /c/estructura-datos (master)
$
```

Ya podemos ver nuestro primer proyecto (Programa) en la nuevo de **GitHub**



## IMPORTANTE

Cada vez que se modifique un proyecto (programa) o se cree uno nuevo, se debe ejecutar del paso 2 ( **Paso 2. Subir nuestros proyectos a GitHub**) los siguientes sub-pasos:

3. Una vez creada las carpetas de cada clase y luego de haber instalado **Git** y **GitHub**, hacer clic derecho sobre la carpeta “*estructura-datos*” y luego sobre la opción “**Git Bash Here**”.
6. Para subir ese primer ejercicio resuelto, debes repetir solo el paso 3 y cuando aparezca el Bash de **GIT** escribir la siguiente línea de comando:



7. Una vez ejecutado el `git add` . ejecutamos la siguiente línea de comando para validar el estado actual de nuestro **Git**.
8. Una vez ejecutado el comando `git status` ejecutamos la siguiente línea de comando para instanciar los cambios preparados en ese momento.
10. Ejecutamos la siguiente línea de comando que tomamos del paso agregar un nuevo control remoto.
11. Finalmente, luego de ejecutar la línea de comando anterior, ejecutamos un `git push` para subir tus cambios locales a tu repositorio en línea.